A Fast and Log-Euclidean Polyaffine Framework for Locally Linear Registration

Vincent Arsigny, Olivier Commowick, Nicholas Ayache, Xavier Pennec

Journal of Mathematical Imaging and Vision manuscript No.
(will be inserted by the editor)

A Fast and Log-Euclidean Polyaffine Framework for Locally Linear Registration

Vincent Arsigny · Olivier Commowick · Nicholas Ayache · Xavier Pennec

Received: date / Accepted: date

Abstract In this article, we focus on the parameterization of non-rigid geometrical deformations with a small number of flexible degrees of freedom. In previous work, we proposed a general framework called *polyaffine* to parameterize deformations with a finite number of rigid or affine components, while guaranteeing the invertibility of global deformations. However, this framework lacks some important properties: the inverse of a polyaffine transformation is not polyaffine in general, and the polyaffine fusion of affine components is not invariant with respect to a change of coordinate system. We present here a novel general framework, called *Log-Euclidean polyaffine*, which overcomes these defects.

We also detail a simple algorithm, the *Fast Polyaffine Transform*, which allows to compute very efficiently Log-Euclidean polyaffine transformations and their inverses on regular grids. The results presented here on real 3D locally affine registration suggest that our novel framework provides a general and efficient way of fusing local rigid or affine deformations into a global invertible transformation without introducing artifacts, independently of the way local deformations are first estimated.

Keywords Locally affine transformations, medical imaging, ODE, diffeomorphisms, polyaffine transformations, Log-Euclidean, non-rigid registration

V. Arsigny, O. Commowick, N. Ayache, X. Pennec
Asclepios project-team,
INRIA Sophia-Antipolis mediterranee,
2004 Route des Lucioles, B.P. 93
F-06902 Sophia Antipolis Cedex, France
E-mail: Vincent.Arsigny@Polytechnique.org
E-mail: Olivier.Commowick@Sophia.inria.fr
E-mail: Nicholas.Ayache@Sophia.inria.fr
E-mail: Xavier.Pennec@sophia.inria.fr

1 Introduction

The registration of medical images is in general a difficult problem, and numerous methods and tools have been already devised to address this task [13]. Still currently, much effort continues to be devoted to finding adequate measures of similarity, relevant parameterizations of geometrical deformations, efficient optimization methods, or realistic mechanical models of deformations, depending on the precise type of registration considered.

In this article, we focus on the parameterization of non-rigid geometrical deformations with a small number of flexible degrees of freedom. This type of parameterization is particularly well-adapted for example to the registration of articulated structures [16] and to the registration of histological slices [18, 1]. After a *global* affine (or rigid) alignment, this sort of parameterization also allows a finer *local* registration with *very smooth* transformations [6, 15, 8, 19].

In [1], we parameterized deformations with a small number of *rigid or affine components*, which can model smoothly a large variety of local deformations. We provided a general framework to fuse these components into a global transformation, called *polyrigid* or *polyaffine*, whose *invertibility* is guaranteed. However, this framework lacks some important properties: the inverse of a polyaffine transformation is not polyaffine in general. The invertibility of transformations is important in registration problems as it ensures that one can resample images and segmented structures. Guarantying invertibility, and more generally the diffeomorphic nature of transformations has been a goal promoted for a long time in the medical image analysis community. Here the point is not really about the invertibility itself, which is already ensured by the previous poly-affine framework, but rather that if the inverse of a polyaffine transformation should remains a polyaffine transformation so that registering one image to the other or the reverse could be considered in the same way.

This is necessary for instance to compute consistent statistics on transformations. A second property which is lacking is that the polyaffine fusion of affine components is not invariant with respect to a change of coordinate system (i.e. is not *affine-invariant*). Here, we present a novel general framework to fuse rigid or affine components, called *Log-Euclidean polyaffine*, which overcomes these defects and yields transformations which can be very efficiently computed.

The sequel of this article is organized as follows. In Section 2, we present the Log-Euclidean polyaffine framework and its intuitive properties. Then, we present the *Fast Polyaffine Transform* (FPT), which allows to compute very efficiently Log-Euclidean polyaffine transformations (LEPTs) and their inverses on a regular grid. Finally, we illustrate how FPTs can be used in a real example of 3D registration based the algorithm of [6].

2 A Log-Euclidean Polyaffine Framework

2.1 Previous Polyaffine Framework

Before presenting our novel polyaffine framework let us briefly recall the original polyaffine framework, described originally in [1]. The idea is to define transformations that exhibit a locally affine behavior, with nice invertibility properties. Following the seminal work of [12], we model here such transformations by a finite number N of affine *components*. Precisely, each component i consists of an affine transformation T_i and of a non-negative *weight function* $w_i(x)$ which models its spatial extension: the influence of the i^{th} component at point x is proportional to $w_i(x)$. Furthermore, we assume that for all x, $\sum_{i=1}^{N} w_i(x) = 1$, i.e. the weights are normalized.

2.1.1 Fusion of Displacements.

In order to obtain a global transformation from several weighted components, the classical approach to fuse the N components simply consists in averaging the associated displacements according to the weights [20]:

$$T(x) = \sum_{i=1}^{N} w_i(x) T_i(x). \tag{1}$$

The transformation obtained using (1) is smooth, but this approach has one major drawback: although each component is invertible, the resulting global transformation is *not invertible* in general. To remedy this, we proposed in [1] to rely on the averaging of some *infinitesimal* displacements associated to each affine component instead. The resulting global transformation is obtained by integrating an Ordinary Differential Equation (ODE), which is computationally more expensive but guarantees its invertibility and also yields a simple form for its inverse. The nice invertibility properties of this approach are illustrated in Fig. 1.

2.1.2 Polyaffine Framework.

The polyaffine approach can be decomposed into three steps:

– **Step 1: Associating Velocity Vectors to Affine Transformations.** For each component i, one defines a family of *velocity vector fields* $V_i(.,s)$ parameterized by s, which is a time parameter varying continuously between 0 and 1. $V_i(.,s)$ satisfy a consistency property with T_i: when integrated between time 0 and 1, they should give back the transformation T. Hence the following definition:

Definition 1 The family of vector fields $V(.,s)$, where s belongs to $[0,1]$, is consistent with the transformation T if and only if its integration between time 0 and 1 gives back the transformation T:

1. for any initial condition x_0 one can integrate between 0 and 1 the differential equation $\dot{x} = V(x,s)$ so that $x(1)$ exists.
2. $x(1)$ is equal to $T(x_0)$.

Several possible choices exist to associate velocity vector to affine transformations. One of the main contributions of this work is precisely to propose a novel choice for such speed vectors. Interestingly, we do not know at present how many other choices exist and whether they might have even better properties than the ones we have found so far.

– **Step 2: Fusing Velocity Vectors instead of Displacements.** The idea is then to average the vector fields $V_i(.,s)$ according to the weight functions $w_i(x)$ to define an ODE fusing the N components.

$$\dot{x} = V(x,s) \stackrel{def}{=} \sum_i w_i(x) V_i(x,s). \tag{2}$$

This ODE is the infinitesimal analogous of the weighted mean of displacements (Eq.1). Weight functions are very important and model the influence in space of each component. They control in particular the sharpness of transitions between the fused affine transformations. Also, they can take into account the geometry of anatomical regions of interest, as will be the case in the experimental results on 3D MRI data given in the sequel.

– **Step3: Integration of the Polyaffine ODE.** In this infinitesimal framework, the value at point x_0 of the global transformation T fusing the N components is obtained via the integration of Eq. (2) between 0 and 1, with the initial condition $x(0) = x_0$.

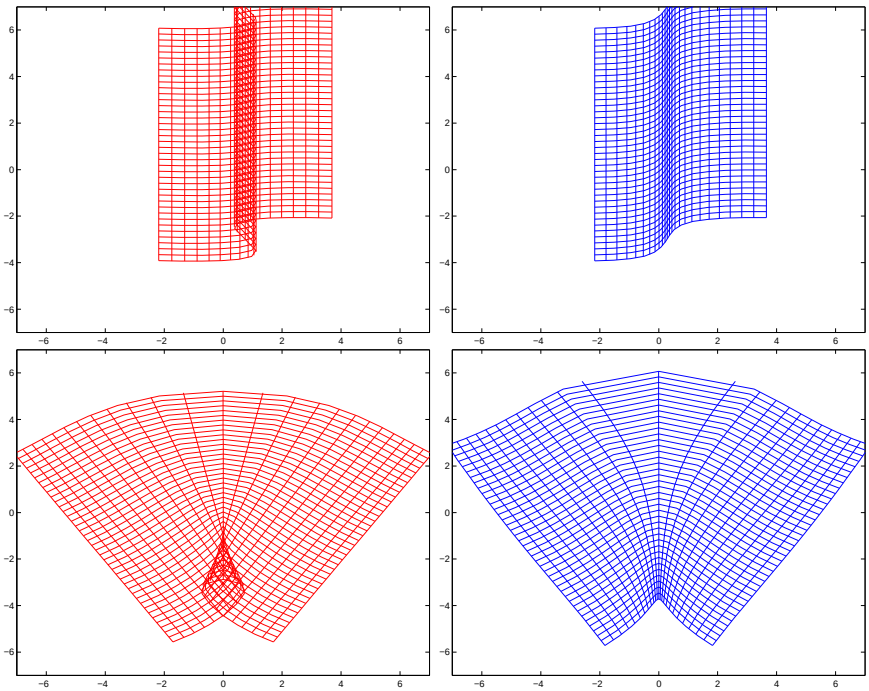

Fig. 1 Example result of the fusion of two affine transformation with the direct and the infinitesimal approaches. Two translations (**on top**) and two rotation (**on bottom**) of opposite axis / angle are fused and we display here the resulting deformation of a regular grid with the direct averaging of displacements **on the left**, and with the infinitesimal fusion of the transformations in the polyaffine framework **on the right**. One can clearly see that the grid folds onto itself with the direct averaging method. This means that different physical points are mapped to the same space point after deformation. Thus, the transformation is not invertible as the points where the grid overlap should be mapped to two different locations. On the contrary, the grid deformed by the infinitesimal method does not fold and remains invertible in this example. The two regions used were centered at points $(-2,0)$ and $(+2,0)$, with the following weights that ensure a smooth transition between the two components (given here in unnormalized form): $w_i(x) = 1/(1+((x_1-c_i)/\sigma)^2)$, where $c_1 = -2$, $c_2 = +2$ and $\sigma = 5$. The affine transformations of the two components where translations $t_1 = (3,1)^T$ and $t_2 = (-1.5,3)^T$ for the top images, and two rotations of opposite angles of magnitude 0.63 radians around the origin of each region for the bottom images.

2.1.3 What Velocity Vectors for Affine Transformations at Step 1?

Let us take an affine transformations $T = (M,t)$, where M is the linear part and t the translation. To define a family of velocity vector fields consistent with T, it was proposed in [1] to rely on the *matrix logarithm* of the linear part M of T. More precisely, let L be the principal matrix logarithm of M. The family of speed vector fields $V(.,s)$ we associated to T writes:

$$V(x,s) = t + L.(x - st) \text{ for } s \in [0,1]. \tag{3}$$

In practice, the matrix logarithm can be efficiently computed using the 'Inverse Scaling and Squaring' method [5]. As for the 'Scaling and Squaring' method in the exponential case, this algorithm is based on the idea that computing the logarithm of a matrix *close to the identity* can be done very accurately and at a very small computational cost, for instance using Padé approximants. In order to transform a matrix into another matrix closer to the identity, the 'Inverse Scaling and Squaring' method uses the computation of successive *square roots*. Once the $2^{N\text{th}}$ root of a matrix M has been computed, one can use the following equality to compute the logarithm of M:

$$\log(M) = 2^N . \log\left(M^{2^{-N}}\right). \tag{4}$$

More details on how square roots can be iteratively computed and on the choice of the level of squarings N can be found in [5].

2.1.4 Well-Definiteness of the Principal Logarithm.

One should note that using principal logarithms of the linear part of affine transformations at the first step of the polyaffine framework is not always possible. The theoretical limitation implied by this particular choice of velocity vectors is the following: the principal logarithm of an invertible matrix M is well-defined when the (complex) eigenvalues of M do not lie on the (closed) half-line of negative real numbers [5]. A more detailed analysis of the necessary and sufficient condition for a real matrix to have a real logarithm is given in [9].

For rotations, this means quite intuitively that the amount of (local) rotation present in each of the components should

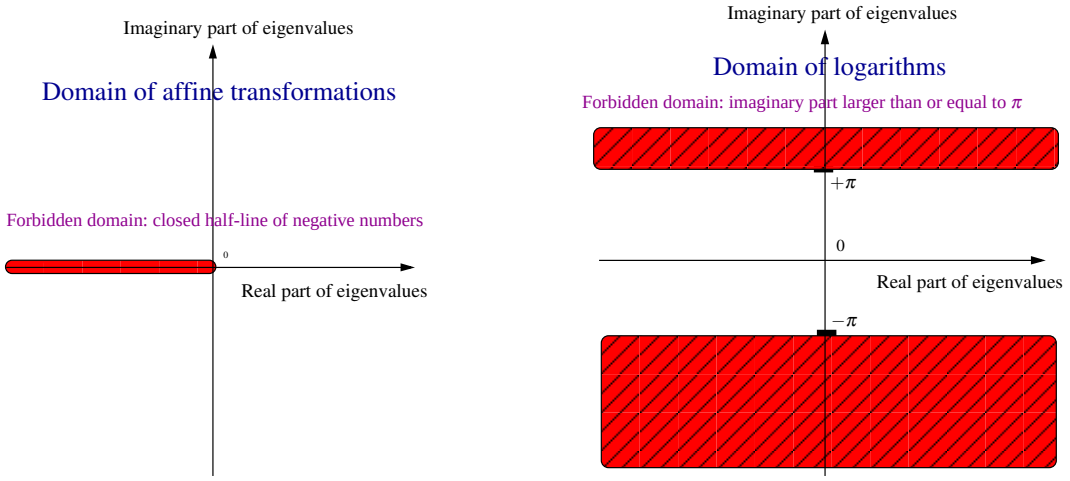

Fig. 2 Constraints imposed on affine transformations by the use of the principal matrix logarithm. Left: only affine transformations whose (complex) eigenvalues do not lie on the (closed) half-line of negative real numbers have a principal logarithm and can be handled by our framework. Intuitively, this corresponds to imposing that (local) rotations be smaller in magnitude than π radians. This can be seen more clearly on the principal logarithms of these *admissible* affine transformations: the imaginary part of their eigenvalues must be smaller than π in magnitude. This is illustrated on the **right part** of this figure. A more detailed discussion of this constraint is given in Subsection 2.1.

be strictly below π radians in magnitude. This can be clearly seen in the domain of matrix logarithms, where this constraint corresponds to imposing that the imaginary part of eigenvalues be less then π in magnitude. Fig. 2 illustrates this general situation, which is not specific to rotations.

For general invertible linear transformations with positive determinant, the interpretation of this constraint on eigenvalues is not so clear, since rotational and non-rotational deformations are intertwined. However, one should note the closed half-line of negative number is a set of null (Lebesgue) measure in the complex plane, which indicates that very few linear transformations with positive determinant (corresponding to extremely large deformations) will not have a principal matrix logarithm. From a practical point of view, one can anyway just check whether the constraint is satisfied by computing numerically the eigenvalues of M, which only amounts to solving a third degree polynomial equation for 3D affine transformations.

In the context of medical image registration, we do not believe this restriction to be problematic, since a global affine alignment of the images to be registered is always performed first. This factors out the largest rotations and it would be very surprising from an anatomical point of view to observe very large deformations (e.g., local rotations close to 180 degrees) of an anatomical structure from one individual to another after the affine alignment of the anatomies of these individuals.

2.1.5 Heavy Computational Burden at Step 3.

Now, from a practical point of view, integrating the ODE given by Eq. (2) with the velocity vectors of Eq. (3) is quite computationally expensive, especially when one wishes to do this for all the points of a 3D regular grid, for example a $256 \times 256 \times 100$ grid, which is commonly in the case for T_1-weighted MR images. We will see in the rest of this section how one can drastically reduce this complexity by slightly modifying the speed vectors of Eq. (3).

2.2 Simpler Velocity Vectors for Affine Transformations

Let us now see how one can define much simpler velocity vectors for affine transformations than the ones given in Eq. (3). The basic idea is to rely on the logarithms of the transformations *themselves*, and not only on the logarithms of their *linear parts*. These logarithms can be defined in an abstract way in the context of the theory of Lie groups, as detailed in [2]. Interestingly, thanks to the faithful representation of these transformations obtained with *homogeneous coordinates*, these logarithms can be computed in practice via matrix logarithms.

2.2.1 Homogeneous Coordinates.

Homogeneous coordinates are a classical tool in Computer Vision. They are widely used to represent any n-dimensional affine transformation T by $(n+1) \times (n+1)$ matrix, written here $\tilde{T}$. Such a representation is called by mathematicians 'faithful' (in the sense of representation theory), which means that there is no loss of information in this representation. $\tilde{T}$ takes the following form:

$$T \sim \tilde{T} \stackrel{def}{=} \begin{pmatrix} M & t \\ 0 & 1 \end{pmatrix}, \tag{5}$$

where M is the linear part of T ($n \times n$ matrix) and t its translation. In this setting, points x of the ambient space are represented by $n+1$-dimensional vectors $\tilde{x}$, adding an extra '1' after their coordinates:

$$x \sim \tilde{x} \stackrel{def}{=} \begin{pmatrix} x \\ 1 \end{pmatrix}.$$

This way, the action of the affine transformation on a point x can be obtained simply in terms of matrix multiplication and is given by $\tilde{T}.\tilde{x}$.

2.2.2 Principal Logarithms of Affine Transformations.

Using homogeneous coordinates, the principal logarithm of the affine transformations *themselves* can be computed in a simple way.

The main point here is that the principal logarithm of an affine transformation T is represented in homogeneous coordinates by the matrix logarithm of its representation $\tilde{T}$. This matrix logarithm takes the following form:

$$\log\left(\tilde{T}\right) = \begin{pmatrix} L & v \\ 0 & 0 \end{pmatrix},$$

where log stands for the principal matrix logarithm. L is an $n \times n$ matrix and v an n-dimensional vector. Exactly as in the former subsection, L is the principal matrix logarithm of M. But v is *not* equal in general to the translation t. Actually, the difference between our novel approach and the previous one resides essentially in this v.

Interestingly, the well-definiteness of the principal logarithm of an affine transformation T is equivalent to the well-definiteness of the principal logarithm of its linear part M. The reason for this is that the spectrum of $\tilde{T}$ is exactly that of its linear part M plus an extra eigenvalue equal to 1, thanks to the form taken by $\tilde{T}$ (see Eq. (5)). Hence the equivalence of the existence of both principal logarithms.

2.2.3 Simpler Velocity Vectors for Affine Transformations at Step 1 of the Polyaffine Framework.

Using now principal logarithms of affine transformations instead of the principal logarithms of their linear parts, one can now associate to an affine transformation T a simpler family of velocity vector fields than in Eq. (3) in the following way:

$$V(x,s) = V(x) = v + L.x \text{ for } s \in [0,1]. \tag{6}$$

What is remarkable here is that the velocity vector field at time s associated to T *does not depend on* s! To prove the consistence of this speed vector with T, let us write the associated ODE:

$$\dot{x} = v + L.x. \tag{7}$$

While the mathematical form taken by (7) might seem unfamiliar, it is much simpler (and more familiar) when expressed in homogeneous coordinates. It simply writes:

$$\dot{\tilde{x}} = \log(\tilde{T}).\tilde{x}, \tag{8}$$

which is this time a *linear* ODE. It is well-known from the theory of linear ODEs [23] that Eq. (8) can be solved analytically and that its solutions are well-defined for all time. With an initial condition x_0 at time 0, the value $x(s)$ of the unique mapping $x(.)$ satisfying Eq. (7) is given in terms of matrix exponential by:

$$\tilde{x}(s) = \exp\left(s.\log\left(\tilde{T}\right)\right).\tilde{x}_0. \tag{9}$$

By letting s be equal to 1, we thus see that our new velocity vectors are truly consistent with the transformation T.

The ODE of Eq. (7) is called *autonomous* (or equivalently *stationary*). Such ODEs have some very nice mathematical properties, which can be expressed in terms of *one-parameter subgroups* of transformations. These properties are detailed in [2, Chapter 2]. In short, the flow $\Phi(.,s)$ of an *autonomous* ODE is a *one-parameter subgroup of the group of diffeomorphisms*, which means the (possibly) large deformations obtained at time 1 result of the composition of a large number of arbitrarily small identical deformations.

2.2.4 One-Parameter Subgroups of Affine Transformations.

From the explicit form taken by the solutions of this ODE (see Eq. (9)), we can see that the associated flow is simply the family of affine transformations $(T^s(.))$, where T^s is the affine transformation represented by $\exp(s.\log(\tilde{T}))$, i.e. the s^{th} power of T.

From the general properties of flows associated to autonomous ODEs, we know that the family of transformations $(T^s(.))$ is a *one-parameter subgroup* of diffeomorphisms. From this point of view, its infinitesimal generator is the vector field $V(x) = v + L.x$. From the viewpoint of the *affine group* (in contrast to diffeomorphisms), (T^s) is also a one-parameter subgroup of affine transformations, whose infinitesimal generator is this time the principal logarithm of T. Interestingly, it can be shown with the classical tools of Lie groups theory that all continuous one-parameter subgroups of affine transformations are of this form [22].

2.3 Log-Euclidean Polyaffine Transformations

2.3.1 An Autonomous ODE for Polyaffine Transformations.

With the velocity vectors defined by Eq. (6), one can define a novel type of polyaffine transformations using the steps 2 and 3 of the Polyaffine framework. In the sequel,

we will refer to these new polyaffine transformations as *Log-Euclidean polyaffine transformations* (or LEPTs). This name comes from our work on diffusion tensors [3], where we have already used principal logarithms to process another type of data.

More precisely, let (M_i, t_i) be N affine transformations, and let (L_i, v_i) be their respective principal logarithms. Then one can fuse them according to the weights $w_i(x)$ with the following ODE, which is this time *autonomous*, i.e. without any influence of the time parameter s in the second member of the equation:

$$\dot{x} = \sum_i w_i(x)\,(v_i + L_i.x)\,. \tag{10}$$

Exactly as in the case of the non-autonomous polyaffine ODE based on Eq. (3), solutions to this novel ODE are well-defined for all time s (i.e. never go infinitely far in a finite time, do not 'blow up'), regardless of the initial condition. The proof is extremely similar (although simpler, in fact) to that given in [1] for the previous polyaffine framework.

Now, we know from the general properties of stationary ODEs (which were presented above) that the flow $T(s,.)$ of this ODE forms a *one-parameter subgroup* of diffeomorphisms: $T(0,.)$ is the identity and $T(r,.) \circ T(s,.) = T(r+s,.)$.

2.3.2 One-Parameter Subgroups of LEPTs.

Exactly like in the affine case, the ODE given by (10) defines not only a one-parameter subgroup of *diffeomorphisms*, it also yields a one-parameter subgroup of *Log-Euclidean polyaffine transformations*. More precisely, a simple change of variable ($s \mapsto \frac{s}{2}$) shows that the flow at time $\frac{1}{2}$, written here $T(\frac{1}{2},.)$, corresponds to a polyaffine transformation whose parameters are the same weights as the original ones, but where the affine transformations have been transformed into their square roots (i.e. their logarithms have been multiplied by $\frac{1}{2}$). Similarly, the flow at time s, $T(s,.)$ corresponds to a polyaffine transformations with identical weights but with the s^{th} power of the original affine transformations.

As a consequence, $T(s,.)$ can be interpreted as the s^{th} power of the Log-Euclidean polyaffine transformation defined by $T(1,.)$. In particular, the inverse of $T(1,.)$ (resp. its square root) is given simply by $T(-1,.)$ (resp. $T(1/2,.)$), which is the polyaffine transformation with identical weights but whose affine transformations have been inverted (resp. have been transformed into their square roots).

One should note that our previous polyaffine transformations do *not* have the same remarkable algebraic properties as Log-Euclidean polyaffine transformations. In our previous framework, the inverse of a polyaffine transformation was not even in general a polyaffine transformation. LEPTs have very intuitive and satisfactory properties, because they are based on a fusion of velocity vectors much better adapted to the algebraic properties of affine transformations than the speed vectors we previously used.

In the next Section, we will see how this specific *algebraic* property of our novel framework can be used to alleviate drastically the computational cost of Step 3 of the polyaffine framework (i.e. the cost of the integration of the polyaffine ODE).

2.3.3 Affine-Invariance of LEPTs.

Contrary to the previous polyaffine framework, our novel Log-Euclidean framework has another sound mathematical property: *affine-invariance*. This means the Log-Euclidean polyaffine fusion of affine transformations is invariant with respect to any affine change of coordinate system. This type of fusion is thus a fusion between *geometric transformations* and not *matrices* since it does not depend at all on the arbitrary choice of coordinate system chosen to represent them.

To see why this is so, let us see how the various ingredients of our framework are affected by a change of coordinate system induced by an affine transformation A. In homogeneous coordinates, these changes are the following:

- a point $\tilde{x}$ becomes $\tilde{y} = \tilde{A}.\tilde{x}$
- a weight function $\tilde{x} \mapsto w_i(\tilde{x})$ becomes $\tilde{y} \mapsto w_i(\tilde{A}^{-1}.\tilde{y})$
- an affine transformation $\tilde{T}_i$ becomes $\tilde{A}.\tilde{T}_i.\tilde{A}^{-1}$.

In our new coordinate system, the Log-Euclidean polyaffine ODE writes in homogeneous coordinates:

$$\dot{\tilde{y}} = \sum_i w_i(\tilde{A}^{-1}.\tilde{y}) \log\left(\tilde{A}.\tilde{T}_i.\tilde{A}^{-1}\right).\tilde{y}. \tag{11}$$

Then, using the property $\log\left(\tilde{A}.\tilde{T}_i.\tilde{A}^{-1}\right) = \tilde{A}.\log\left(\tilde{T}_i\right).\tilde{A}^{-1}$, the simple change of variable $\tilde{y} \mapsto \tilde{A}.\tilde{x}$ shows that a mapping $s \mapsto \tilde{x}(s)$ is a solution of the Log-Euclidean polyaffine ODE (10) if and only if $s \mapsto \tilde{A}.\tilde{x}(s)$ is a solution of (11). This means that the solutions of the Log-Euclidean polyaffine ODE in the new coordinate system are exactly the same as in the original coordinate system: our novel polyaffine framework is therefore not influenced by the choice of a coordinate system. The previous polyaffine framework does not have this property, because it does not take sufficiently into account the algebraic properties of affine transformations.

2.3.4 Another Reason Why our Novel Polyaffine Framework is Called Log-Euclidean.

In the special case where none of the weight functions $w_i(x)$ depend on x, the Log-Euclidean polyaffine fusion of the affine transformation T_i simply yields an affine transformations T, which is given by the following *Log-Euclidean mean*:

$$T = \exp\left(\sum_i w_i \log(T_i)\right).$$

This is another reason why we refer to our novel polyaffine framework as Log-Euclidean. Indeed, the use of a generalization to rigid and affine transformations of our Log-Euclidean framework for tensors [4] is implicit in this novel framework. More details on the Log-Euclidean framework for linear transformations can be found in [2, Chapter 6].

2.3.5 Synthetic Examples.

Examples of 2D LEPTs are shown in Figs. 3. In these examples, one can see how antagonistic affine transformations (i.e. transformations whose direct fusion results in local singularities) can be globally fused into a regular and invertible polyaffine transformation.

Interestingly, we have observed in our experiments that the Log-Euclidean and the previous polyaffine frameworks provide similar results. Fig. 4 illustrates the striking closeness between both frameworks. Notable differences only appear when very large deformations are fused.

Therefore, the advantage of our Log-Euclidean polyaffine framework over the previous one does not reside in the quality of its results, which are very close to those of the previous one. Rather, it resides in its much better and more intuitive mathematical properties, which allow for much faster computations, as will be shown in the next Section. This situation is somehow comparable to the closeness between the affine-invariant [17] and Log-Euclidean [3] Riemannian frameworks used to process diffusion tensors. They also yield very similar results, but in a simpler and faster way in the Log-Euclidean case.

3 Fast Polyaffine Transform

In this Section, we show how one can use the specific *algebraic* properties of the Log-Euclidean polyaffine framework to obtain fast computations of LEPTs. In particular, we propose an efficient algorithm to evaluate a Log-Euclidean polyaffine transformations on a regular grid. If N is the number of intermediate points chosen to discretize the continuous trajectory of each point, we present here an algorithm only requiring $\log_2(N)$ steps to integrate our autonomous polyaffine ODE, provided that the trajectories of all the points of the regular grid are computed simultaneously. This drastic drop in complexity is somehow comparable to that achieved by the 'Fast Fourier Transform' in its domain. The key to this approach lies in the generalization to the non-linear case of a popular method which is widely used to compute numerically the exponential of a square matrix.

3.1 Matrix Exponential and the 'Scaling and Squaring' Method

The matrix exponential of a square matrix can be computed numerically in a large number of ways, with more or less efficiency [14]. One of the most popular of these numerical recipes is called the 'Scaling and Squaring' method, which is for example used by Matlab ™ to compute matrix exponentials [10]. Fundamentally, this method is very efficient because it takes advantage of the very specific algebraic properties of matrix exponential, which are in fact quite simple, as we shall see now. For any square matrix M, we have:

$$\exp(M) = \exp\left(\frac{M}{2}\right).\exp\left(\frac{M}{2}\right) = \exp\left(\frac{M}{2}\right)^2. \qquad (12)$$

This comes from the fact that M commutes with itself in the sense of matrix multiplication. Iterating this equality, we get for any positive integer N:

$$\exp(M) = \exp\left(\frac{M}{2^N}\right)^{2^N}, \qquad (13)$$

Then, the key idea is to realize that the matrix exponential is much simpler to compute for matrices *close to zero*. In this situation, one can for example use just a few terms of the infinite series of exponential, since high-order terms will be completely negligible. An even better idea is to use Padé approximants, which provide excellent approximations by rational fractions of the exponential around zero with very few terms. For more (and recent) details on this topic, see [10].

The 'Scaling and Squaring' Method for computing the matrix exponential of a square matrix M can be sketched as follows:

1. **Scaling step**: divide M by a factor 2^N, so that $\frac{M}{2^N}$ is close enough to zero (according to some criterion based on the level of accuracy desired: see [10] for more details).
2. **Exponentiation step**: $\exp\left(\frac{M}{2^N}\right)$ is computed with a high accuracy using for example a Padé approximant.
3. **Squaring step**: using Eq. (13), $\exp\left(\frac{M}{2^N}\right)$ is squared N times (only N matrix multiplications are required.) to obtain a very accurate estimation of $\exp(M)$.

In the rest of this Section, we will see how one can generalize this method to compute with an excellent accuracy polyaffine transformations based on autonomous ODEs.

3.2 A 'Scaling and Squaring' Method for LEPTs

The goal of the method described below is to compute efficiently and with a good accuracy the values of a Log-Euclidean polyaffine transformation at the vertices of a regular n-dimensional

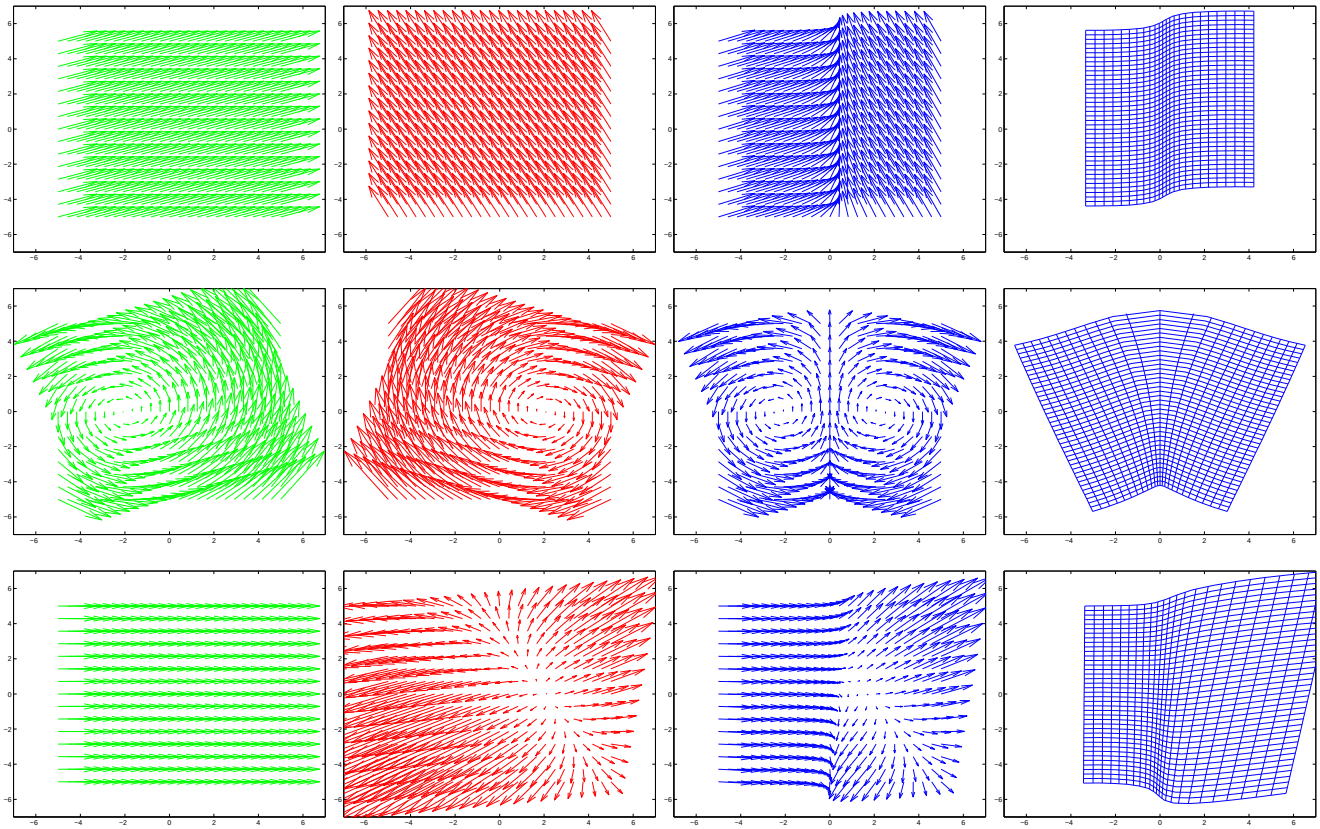

Fig. 3 Fusing velocity vectors of two translations (top row), two rotations (middle row) and a translation and an anisotropic swelling (bottom row). From left to right: Log-Euclidean polyaffine speed vectors computed from the two affine transformations with the novel framework, fused speed vectors, and a regular grid deformed after integration of the autonomous ODE. The weights used for the fusion were two functions of the first coordinate as in Fig. 1. Note how locally antagonistic displacements are fused in an invertible way, resulting in compressions or swelling at the boundary between the two components.

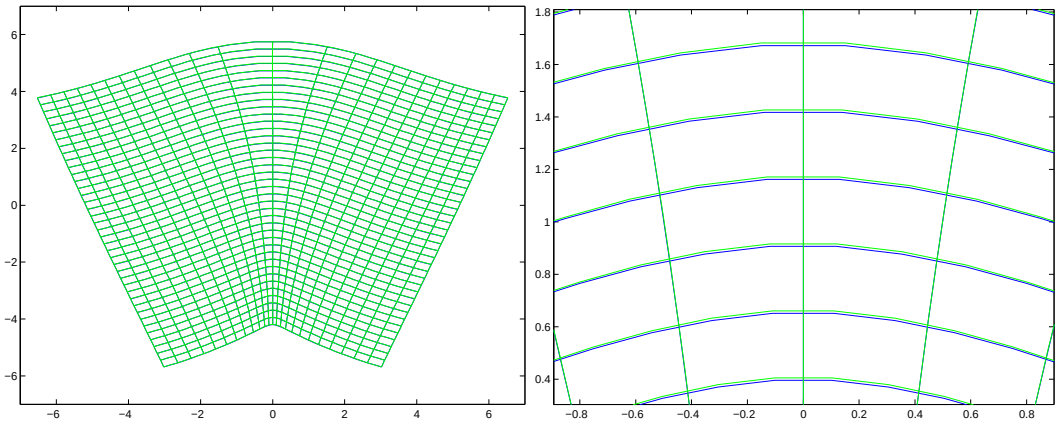

Fig. 4 Comparison between the new and the previous Log-Euclidean polyaffine framework. We superimpose here the deformation of a regular grid resulting from the fusion of two rotations (as in Fig. 3) using the new log-Euclidean method and the previous poly-affine framework. One cannot see any difference at a large scale (on the left): one has to zoom quite intensively to see the very small difference (on the right).

(well, 2D or 3D in practice) grid. In the sequel, this method will be referred to as the 'Fast Polyaffine Transform' (or FPT).

3.2.1 Algebraic Properties of Log-Euclidean Polyaffine Transformations Revisited.

Let $T(s,.)$ be the flow associated to the autonomous polyaffine ODE (10), as in Subsection 2.3. As mentioned before, this flow is a one-parameter subgroup of LEPTs:

$$T(0,.) = Id \quad \text{and for all } r,s\text{: } T(r,.) \circ T(s,.) = T(r+s,.).$$

As a consequence, Exactly as Eq. (12) for the matrix exponential, we obtain for $r = s = \frac{1}{2}$:

$$T(1,.) = T\left(\frac{1}{2},.\right) \circ T\left(\frac{1}{2},.\right) = T\left(\frac{1}{2},.\right)^2.$$

Iterating this equality, we get for any positive integer N:

$$T(1,.) = T\left(\frac{1}{2^N},.\right)^{2^N}. \tag{14}$$

Intuitively, Eq. (14) means that what the deformation observed at time 1 results of 2^N times the repetition of the small deformations observed at time $\frac{1}{2^N}$. The total deformation is entirely determined by the initial (and small) deformations occurring just at the beginning of the integration of our ODE (which is a well-known and general phenomenon with autonomous ODEs).

3.2.2 Fast Polyaffine Transform.

We can now generalize the 'Scaling and Squaring' Method to the Log-Euclidean polyaffine case. This method, called the 'Fast Polyaffine Transform', follows the usual three steps:

1. **Scaling step**: divide $V(x)$ (the field of velocity vectors) by a factor 2^N, so that $\frac{V(x)}{2^N}$ is close enough to zero (according to the level of accuracy desired).
2. **Exponentiation step**: $T\left(\frac{1}{2^N},.\right)$ is computed using an adequate numerical scheme.
3. **Squaring step**: using Eq. (14), $T\left(\frac{1}{2^N},.\right)$ is squared N times (in the sense of the composition of transformations; only N compositions are required) to obtain an accurate estimation of $T(1,.)$, i.e. of the polyaffine transformation to be computed (e.g., an average relative error of the order of 0.5%).

From a practical (or numerical) point of view, two points remain to be clarified. First what numerical scheme can be used to compute $T\left(\frac{1}{2^N},.\right)$ with a good precision during the 'exponentiation step'? Second, how should the composition (which is the multiplication operator for transformations) be performed during the 'squaring step'?

3.2.3 Exponentiation Step.

Exactly as in the matrix exponential case, integrating an ODE during a very short interval of time (short with respect to the smoothness of the solution) is quite easy. We can use any of the methods classically used to integrate ODEs during short periods of times, like explicit schemes or Runge-Kutta methods, which are based on various uses of the Taylor development to compute solutions of ODEs (see [11] for more details on these methods).

The simplest of these schemes is undoubtedly the first-order explicit scheme. In our case, it simply consists in computing the following value:

First Order Explicit Exponentiation Scheme (E.S):

$$T\left(\frac{1}{2^N},x\right)_{\text{E.S.}} \stackrel{def}{=} x + \frac{1}{2^N}.V(x).$$

Generalizing the ideas already developed in [1] for the previous polyaffine framework, we can also use a second-order scheme which takes into account the affine nature of all components, and which is *exact* in the case of a single component. We will refer to this scheme as the *affine exponentiation scheme* in the following. It writes:

Second Order Affine Exponentiation Scheme (A.S):

$$T\left(\frac{1}{2^N},x\right)_{\text{A.S.}} \stackrel{def}{=} \sum_{i=1}^{N} w_i(x).T_i^{\frac{1}{2^N}}(x),$$

where $T_i^{\frac{1}{2^N}}$ is the 2^Nth root of the affine transformation T_i. We will see later in this Section that the accuracy of this numerical scheme is slightly better than that of the explicit scheme, probably because it takes into account the linear nature of the components.

3.2.4 Composing Discrete Transformations.

In this work, we are evaluating our transformation at a *finite* number of vertices of a regular grid. Practically, one has to resort to some kind of interpolation/extrapolation technique to calculate the value of such a transformation at *any* spatial position. Numerous possibilities exist in this domain, such as nearest-neighbor interpolation, bi- or tri-linear interpolation, continuous representations via the use of a basis of smooth functions like wavelets, radial basis functions... In the following, we use bi- and tri-linear interpolations, which are simple tools guaranteeing a continuous interpolation of our transformation. The best type of interpolation technique for the purposes of our Fast Polyaffine Transform remains to be determined and will be the subject of future work.

3.2.5 Algorithmic Complexity.

Note that to compute polyaffine transformations using the FPT, the weight functions need only be evaluated *once* per voxel, and not at *every step* of the integration of the ODE, as was done in [1]. When weight functions are stored in the computer memory as 3D scalar images, this offers the opportunity of removing them from the computer RAM after the exponentiation step. This could be particularly useful when a large number of affine components are used on high-resolution images.

Furthermore, the equivalent of 2^N intermediate points is achieved in only N steps, in contrast with the 2^N steps required by a traditional method. After the 2^Nth root has been computed, only N compositions between transformations need to be computed, which is an operation based on interpolation techniques and therefore not very computationally expensive. Let N_{vox} be the number of voxels and let N_{pts} be the number of intermediary points chosen to integrate the polyaffine ODE. The complexity of our new algorithm is thus $O(N_{\text{vox}}.\log_2(N_{\text{pts}}))$, whereas the complexity of traditional methods of integration of this ODE is $O(N_{\text{vox}}.N_{\text{pts}})$.

3.2.6 Computing the Inverse of a Polyaffine Transformation.

As pointed out in Subsection 2.3, in our new framework the inverse of a polyaffine transformation is simply the polyaffine transformation associated with the opposite vector field (i.e. the polyaffine transformation with the same weights but inverted affine components). As a consequence, the inverse of a polyaffine transformation can be also computed using the Fast Polyaffine Transform. Actually, any power (square root, etc.) a polyaffine transformation can be computed this way.

3.3 2D Synthetic Experiments

Throughout this results Section, we measure the accuracy of our results by computing the relative difference of the results with respect to accurate estimations of the real (continuous) transformations. These reference transformations are obtained by a classical integration (i.e., a fixed time step was used) of the Log-Euclidean polyaffine ODE for each of the pixels of the grid, using a small time step: 2^{-8}.

One should note that several parameters influence the accuracy of the results:

- the scaling 2^N
- the geometry of the regular grid
- the interpolation method
- the extrapolation method.

Thus, compared to the classical estimation method with a fixed time step, our fast transform possesses three new sources of numerical errors: the geometry of the regular grid (the transformation is evaluated only at a finite number of points, the more points the more precise the result will be), the interpolation method and the fact that regardless of the extrapolation method, some part of the information about what happens outside of the regular grid is lost. It is therefore important to check that the accuracy of the results obtained with the FPT are not spoiled by these new sources of error.

3.3.1 A Typical FPT.

Figs. 5 and 6 display the results of a typical Fast Polyaffine Transform, using two rotations of opposite angles, and a scaling of 2^6 (and therefore 6 squarings). The regular grid chosen to sample the transformation is of 50×40 pixels. The affine exponentiation scheme is used.

On average, the results are quite good: the average relative error is approximately equal to 0.6%. However, much higher errors (around 11%) are obtained at the boundary, which comes from the fact that the bi-linear interpolation we use here does not take into account the rotational behavior of the transformation outside of the grid.

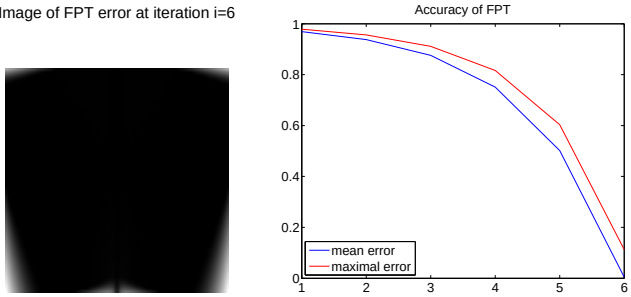

Fig. 6 Fast polyaffine transform for two rotations: error localization and evolution. Left: the errors at the vertices of our 50×40 regular grid are displayed as an image, after a FPT with 6 squarings. The maximal relative errors are concentrated on the boundary of our grid. This is due to the inaccuracy of our extrapolation technique, which is only bi-linear and does not deal very precisely with the affine nature of the polyaffine transformation. **Right:** the evolution of errors along squarings is displayed. The relative error of the resulting estimation of the polyaffine transformation is below 0.6% on average and the maximal relative error is below 11%.

3.3.2 Using Bounding Boxes to Correct Boundary Effects.

The numerical errors stemming from the loss of information at the boundary of the regular grid can be drastically reduced for example by enlarging the regular grid used. A simple idea consists in adding to the regular grid some extra points so that it contains the points of boundary deformed by Euclidean fusion of the affine components. A simple method to compute the bounding box is to rely on the direction fusion of the vector speed, as illustrated by Fig. 7.

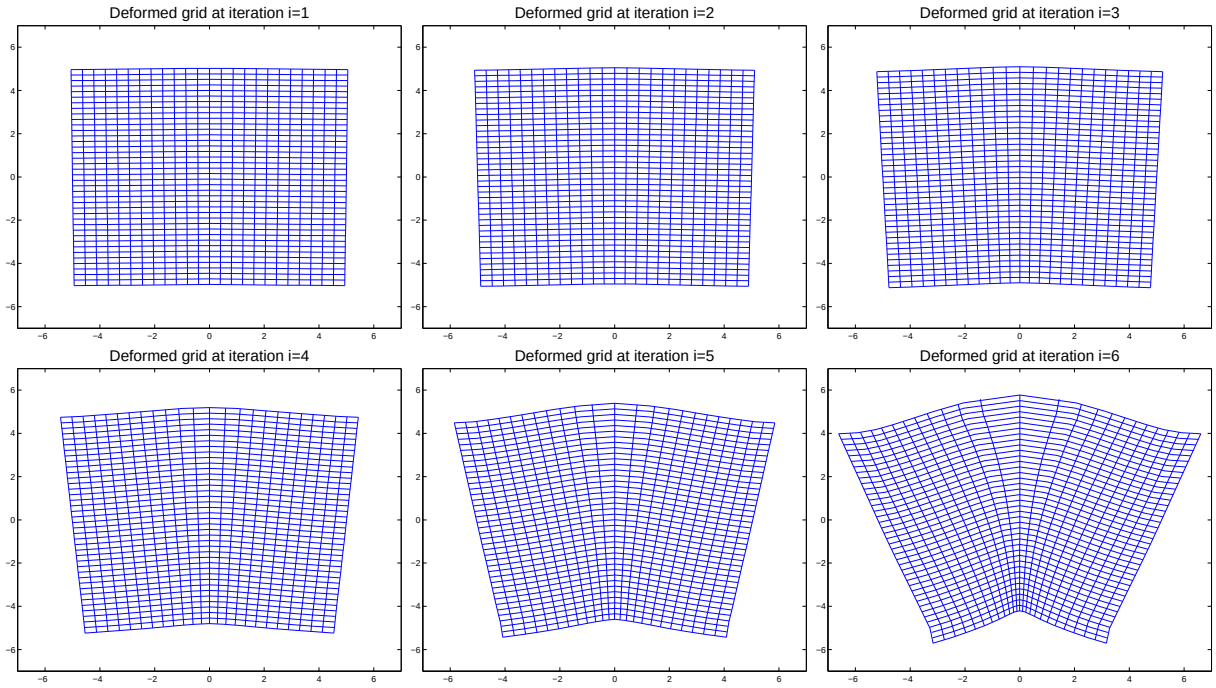

Fig. 5 Fast polyaffine transform for two rotations. A scaling factor of 2^6 was used in this experiment, and there are therefore 6 squaring steps. Note how the deformation is initially very small, and increases exponentially. The accuracy of the FPT results was measured with respect to the results given by a classical integration (voxel by voxel) of the polyaffine ODE with 2^8 intermediate points. The relative error of the resulting estimation of the polyaffine transformation is below 0.6% on average and the maximal relative error, as expected, is made at the boundary and is below 11%.

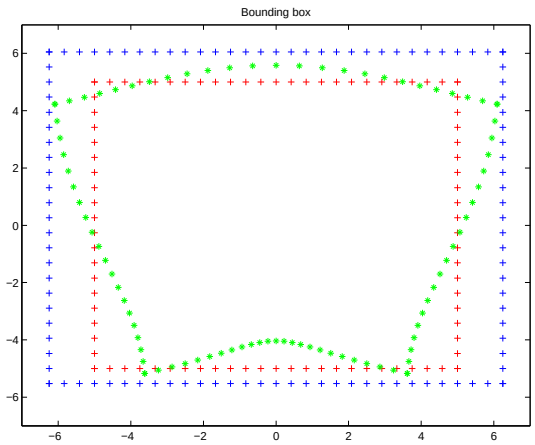

Fig. 7 An extended grid for accurate discrete polyaffine transforms. Internal plus signs: the bounding box of the original regular grid used to sample the Log-Euclidean polyaffine fusion between two rotations. **Stars:** its deformation by the direct fusion of the two rotations, which can be computed at a very low computational cost. **External plus signs:** bounding box of the extended regular grid which now contains all the star points. This enlarging procedure considerably reduces the impact on the Fast Polyaffine Transform of the loss of information beyond the boundaries of the regular grid, as shown in Fig. 8.

Fig. 8 presents the accuracy of the results given by the FPT using such an extended regular grid. This time, errors are much lower: the relative accuracy of the resulting estimation of the polyaffine transformation is on average of 0.21% (instead of approximately 0.6% previously), and the maximal relative error is below 3.2% (instead of 11% previously). This simple and efficient technique, which drastically reduces the effect of boundary effects on the FPT, is used systematically in the sequel.

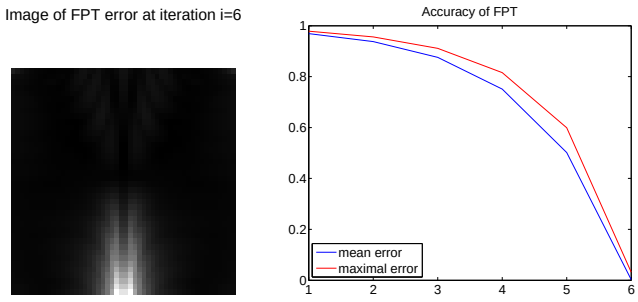

Fig. 8 Using an enlarged sampling grid: impact on errors localization and evolution for the fast polyaffine transform for two rotations. Left: the errors at the vertices of our 50×40 regular grid are displayed as an image, after a FPT with 6 squarings. Note how the maximal errors are concentrated this time on the region of highest compression. **Right:** the evolution of errors along squarings is displayed. This time, errors are much lower: the relative error of the resulting estimation of the polyaffine transformation is on average below 0.21% (instead of below 0.6% without an enlarged grid), and the maximal relative error is below 3.2% (instead of 11% wihtout an enlarged grid).

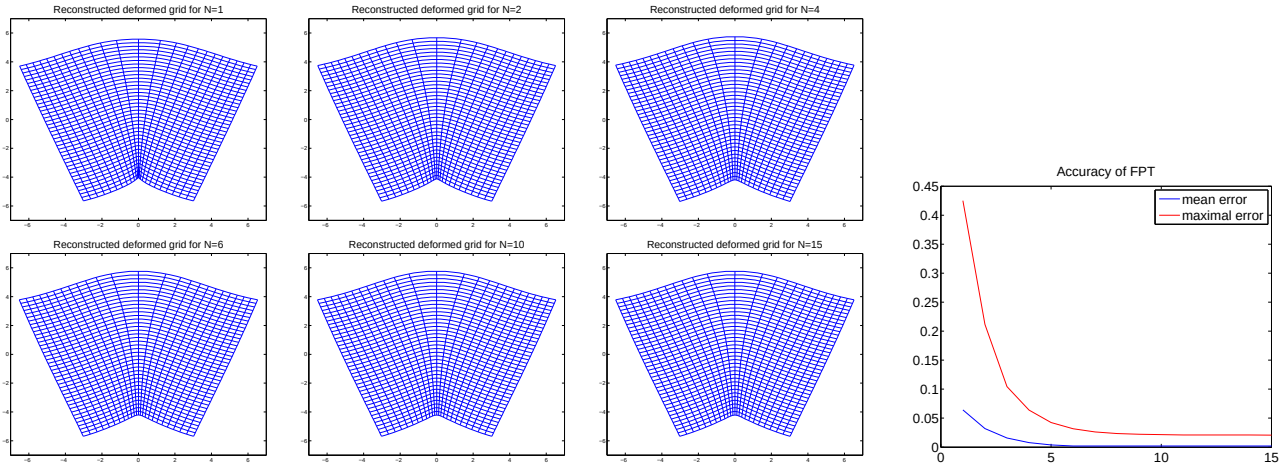

Fig. 9 Fast polyaffine transform for two rotations: influence of the scaling. Left: regular grids deformed by polyaffine transformations obtained with the FPT using scaling factors of (from left to right on top): 2^1, 2^2, 2^4, and (from left to right on bottom): 2^6, 2^{10} and 2^{15}. Note how close the results are when the number of squarings $N > 6$. **Right:** One can see that the accuracy of the results is extremely *stable* for $N > 6$: it is not necessary to use larger scalings in the example considered here. Even more remarkably, using larger scalings *does not degrades* the results: our FPT is very stable.

3.3.3 Influence of Scaling.

What scaling should be chosen when the FPT is used? Of course, this depends on the quantity of high frequencies present in the polyaffine transformations. The more sharp changes, the smaller the scaling should be and the finer the sampling grid should also be.

Fig. 9 displays the performance in accuracy of the FPT when the number of iterations N varies. In this experiment, we use the same fusion of rotations as in the previous experiment. In this case, the optimal scaling is 2^5. Larger scalings do not result in better accuracy, essentially because of the missing information at the boundary.

We observed in the experiments on real 3D medical images described in the sequel that even much smaller scalings (typically 2^3 or 2^2) could be used without sacrificing the accuracy of the result. In short, introducing even a small number of intermediate points substantially regularizes the fused transformation with respect to the direct fusion, since this suffices to remove singularities in practice. Using more intermediary points, i.e. 5 or more squarings, offers the possibilities to be very close to the ideal polyaffine transformation, which provides a simple way to compute the inverse of the fused transformation with an excellent accuracy, as will be shown in this subsection.

Moreover, one should also note from Fig. 9 that our Fast Polyaffine Transform is very *stable*: using unnecessary iterations (or equivalently a very large scaling) does not result in numerical instabilities. The result is mostly independent of N for $N > 6$. The relative error of the resulting estimation of the polyaffine transformation converges toward 0.2% on average and the maximal relative error converges toward 2% for large Ns. The residual maximal error is essentially due to the sampling of the transformation on a grid and the use of an interpolation method between the points of the grid, since an extended grid is used to drastically reduce errors at the boundary of the grid.

3.3.4 Comparison between Numerical Schemes.

Here, we compare the explicit affine exponentiation schemes. We perform this comparison on our three favorite examples: the fusion of two rotations, the fusion of two antagonistic translations, and the fusion between a translation and an anisotropic swelling as in Fig. 3. The accuracy of the FPT using both numerical schemes is compared in all three cases. Fig. 10 shows the results.

Both numerical schemes make the FPT converge toward the same accuracy as the number of squarings increases, but the convergence is slightly faster in the affine exponentiation case: the average error is 40% smaller in the affine case for scalings smaller than 2^6. Interestingly, the two numerical schemes are identical for the fusion of the two translations, because the linear parts of these two affine translations are equal to the identity.

3.3.5 Inverting Polyaffine Transformations with the FPT.

As pointed out previously, in our novel framework, the inverse of a polyaffine transformation is simply (and quite intuitively) the polyaffine transformation with the same weights and with inverted affine components. This inverse can also be computed using the Fast Polyaffine Transform, and in this experiment we tested the accuracy of the inversion obtained this way. The affine exponentiation scheme was used for exponentiation along with a 50×40 grid.

Fig. 11 presents with deformed grids the evolution of the accuracy of inversion when the number of squarings varies, in our example of fusion between two rotations. Fig. 12 presents the quantitative results in the three cases of fusion used in the previous experiment. We thus see that an excellent quality of inversion can be achieved using a small number of squarings, typically 6.

3.4 3D Registration Example

In [1] we showed how it is possible to optimize the parameters of polyrigid or polyaffine transformations in medical image registration experiments. However, this leads in practice to a high computational cost. To obtain short computation times (typically 10 minutes for whole 3D volumes in the locally affine case), we proposed in [6, 7] a multi-resolution and robust block-matching scheme for the locally affine registration of multiple components. This algorithm estimates affine components using the *direct fusion*. The FPT is used in a *final step* to ensure the invertibility of the final transformation, as well as to compute its inverse. We have observed experimentally that this yields quite satisfactory results, as we will see below.

Let us consider a real 3D example: the registration, of an atlas of $216 \times 180 \times 180$ voxels to a T_1-weighted MR image. Seven structures of interest are considered to define the locally affine transformation: eyes (1 affine component each), cerebellum (2 components), brain stem (2 components), optic chiasm (1 component), 1 supplementary component (set to the identity) elsewhere. Weight functions are defined in the atlas geometry using mathematical morphology and a smoothing kernel in a preliminary step and remain unchanged during the registration process.

Philosophy of our Locally Affine Algorithm. The idea is to register finely our structures of interest, with *very smooth* local transformations. In contrast, many registration algorithms are able to register *the intensities* all over the images of two anatomies, but this is done in most cases at the cost of the regularity of the resulting spatial transformation. This lack of smoothness leads to serious doubts regarding the anatomical likelihood of such transformations.

Fig. 13 provides a comparison between the typical smoothness of dense transformation and locally affine registration results. Interestingly, much smoother deformations are obtained in the locally affine case with an accuracy in the structures of interest which is comparable to the dense transformation case of [21]. More details on this subject can be found in [7].

LEPTs as a Powerful Post-Processing Tool. As we mentioned before, our locally affine registration algorithm estimates affine components using the *direct fusion*. The FPT is used in a *final step* to ensure the invertibility of the final transformation, as well as to compute its inverse. Here, the scaling used in 2^8 and the FPT is computed in 40*s* on a Pentium4 Xeon^{TM}2.8 GHz on a $216 \times 180 \times 180$ regular grid.

As shown by Fig. 14, the direct fusion of components estimated by our locally affine algorithm can lead to singularities, which is not the case when the FPT is used. Re-

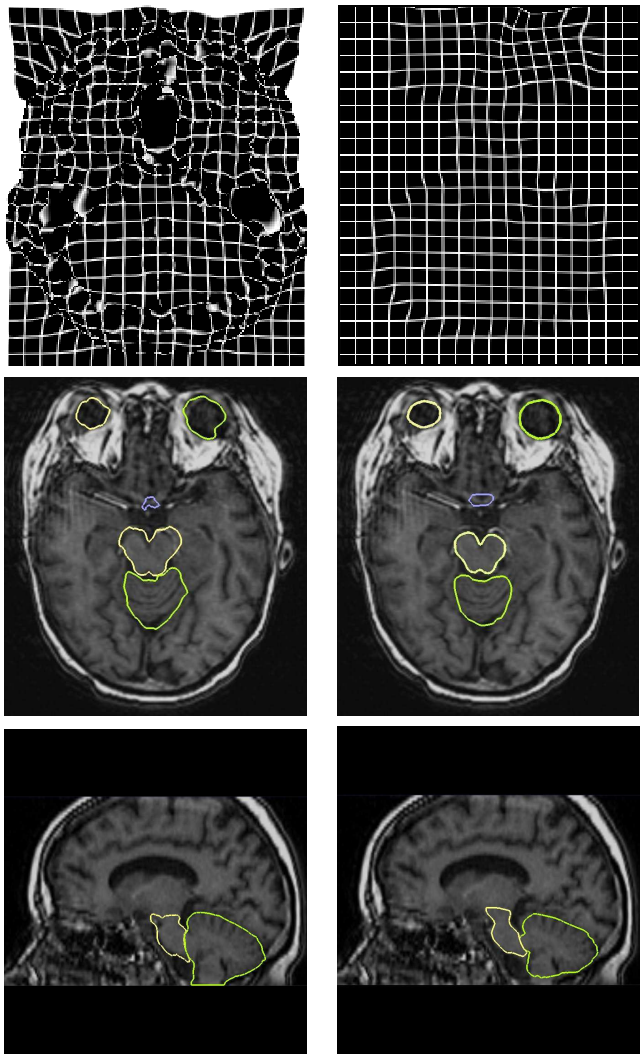

Fig. 13 Comparison of the smoothness of deformations for our locally affine vs. a dense deformation algorithm. An atlas containing a MR T1 image of a reference subject and the segmentation of structures of interest (eyes, brain stem, cerebellum, optic chiasm) is registered to the MR T1 image of a patient using a dense transformation algorithm [21] **on the left** and the locally affine registration algorithm based on the fast polyaffine framework proposed in [7] **on the right**. We display **on top:** the deformation of a regular grid by both transformation on an axial slice. One can clearly see that there are locally very large deformations with the dense deformation algorithm while the deformation remains globally very smooth with the locally affine algorithm. **Middle and bottom:** we display the contours of the structures of interest of the atlas deformed into the geometry of the patient image. The contours appear to be smoother in the locally affine case, although the accuracy is comparable with both methods.

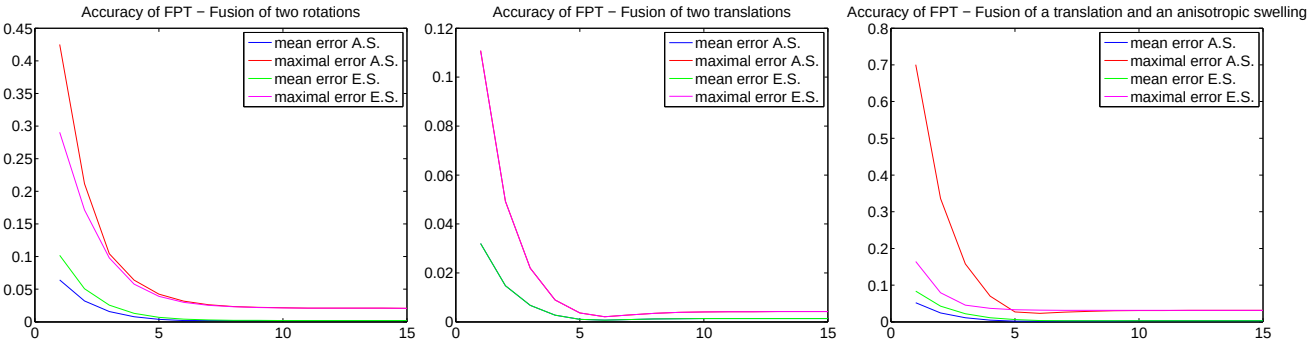

Fig. 10 Comparison between numerical schemes. From left to right and then from top to bottom: fusion between two rotations, two translation and finally a translation and an anisotropic swelling. A.S. stands for 'affine exponentiation scheme' and E.S. for 'explicit exponentiation scheme'. Interestingly, the two numerical schemes are identical for the fusion of the two translations, because the linear parts of these two affine translations are equal to the identity. Both numerical schemes make the FPT converge toward the same accuracy as the number of squarings increases, but the convergence is substantially faster in the case of the affine exponentiation scheme: in the two cases where the schemes yield different results, the average relative error is 40% smaller in the affine case for scalings smaller than 2^6.

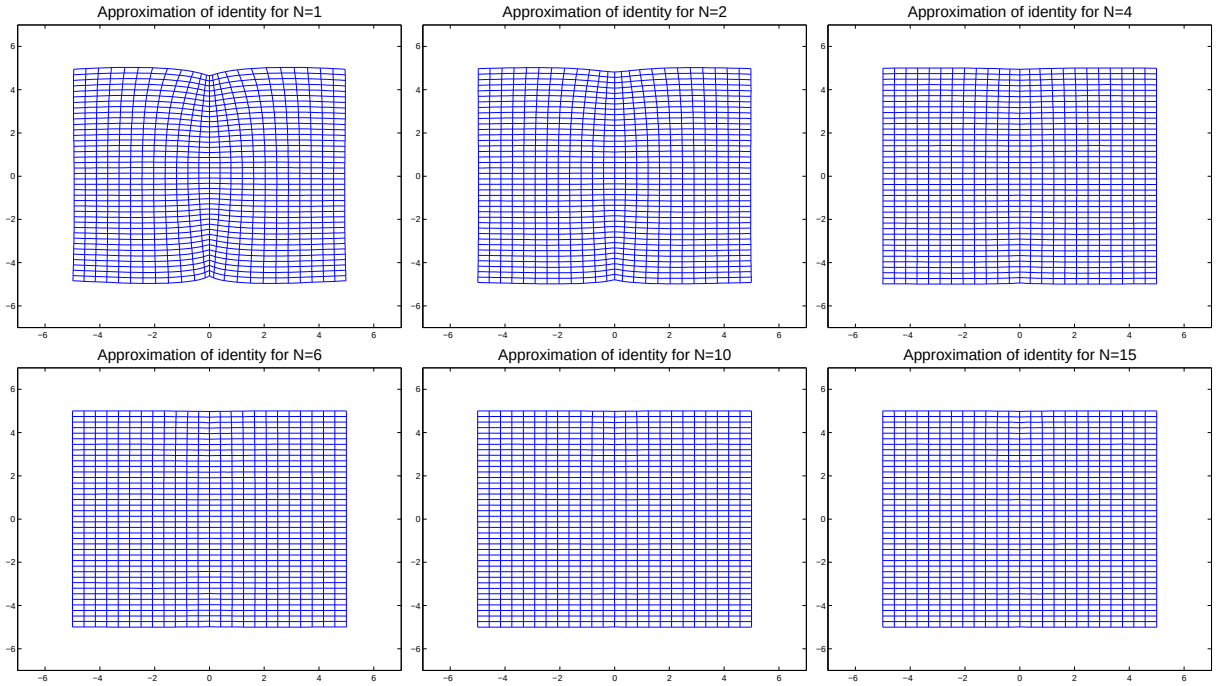

Fig. 11 Inverting a polyaffine transformation with the FPT. From left to right and then from top to bottom: our regular grid is deformed by the composition between the FPT of the fusion between two rotations and the FPT of its inverse, for different numbers of squarings N. One can see that an excellent accuracy of inversion is already achieved with 6 squarings.

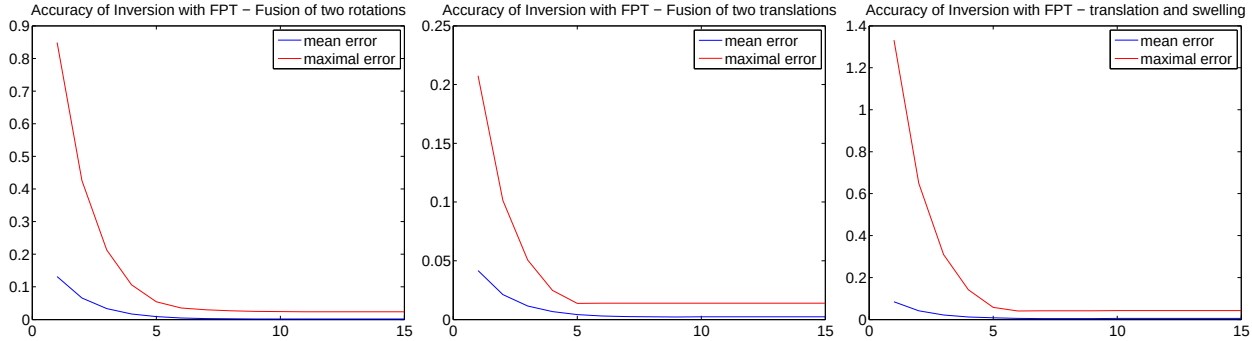

Fig. 12 Inverting a polyaffine transformation with the FPT: quantitative results. From left to right and then from top to bottom: fusion between two rotations, two translation and finally a translation and an anisotropic swelling. The composition between the FPT of the transformation of the FPT of its inverse is carried out, for different numbers of squarings. The errors displayed are relative with respect the polyaffine transformation considered: the displacements are expected to be close to zero (i.e. the resulting transformation is expected to be close to the identity), and the errors are measured with respect to the displacements observed originally. One can see that an excellent accuracy of inversion is already achieved with 6 squarings. As expected, the maximal errors are observed at the boundary of the grid, which can be fixed for example by using a larger grid to compute the FPT.

markably, both fusions are very close *outside* of regions with singularities. This means that no artifacts are introduced by the FPT, which justifies *a posteriori* the estimation of affine components with the (faster) direct fusion.

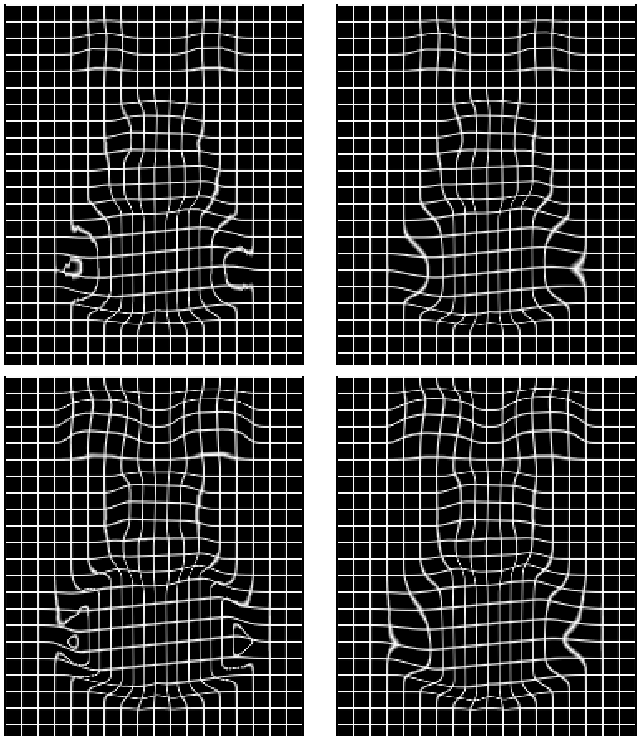

Fig. 14 Singularity removal with LEPTs. A 3D regular grid is deformed with the locally affine transformation obtained with the algorithm of [6,7], two different axial slices are displayed on top and bottom. **On the left:** result of the direct fusion. One can notice small loops that locally disrupt the regular structure of the grid at the boundary of two regions of influence. They indicate places where the deformation is not diffeomorphic. **On the right:** result of the polyaffine fusion. These singularities of the direct fusion disappeared with LEPTs, without modifying much the deformation outside the regions where the singularities were located.

4 Conclusion and Perspectives

In this work, we have presented a novel framework to fuse rigid or affine components into a global transformation, called *Log-Euclidean polyaffine*. Similarly to the previous polyaffine framework of [1], it guarantees the *invertibility* of the result. However, contrary to the previous framework, this is achieved with very intuitive properties: for example the inverse of a LEPT is a LEPT with identical weights and inverted affine components. Moreover, this novel fusion is *affine-invariant*, i.e. does not depend on the choice of coordinate system. We have also shown that remarkably, and contrary to previous polyaffine transformations, the specific properties of LEPTs allow their fast computations on regular grids, with an algorithm called the 'Fast Polyaffine Transform', whose efficiency is somehow comparable to that of the Fast Fourier Transform.

In the example of locally affine 3D registration presented here, we use LEPTs in a final step to fuse the affine components estimated during the algorithm of [6]. With the FPT, this is done very efficiently. Remarkably, the novel fusion is *very close* to the direct fusion in regions without singularities. This suggests that our novel framework provides a general and efficient way of fusing local rigid or affine deformations into a global invertible transformation without introducing artifacts, *independently* of the way local affine deformations are first estimated.

References

1. V. Arsigny, X. Pennec, and N. Ayache. Polyrigid and polyaffine transformations: a novel geometrical tool to deal with non-rigid deformations - application to the registration of histological slices. *Medical Image Analysis*, 9(6):507–523, December 2005.
2. Vincent Arsigny. *Processing Data in Lie Groups: An Algebraic Approach. Application to Non-Linear Registration and Diffusion Tensor MRI*. Thèse de sciences (PhD Thesis), École polytechnique, November 2006.
3. Vincent Arsigny, Pierre Fillard, Xavier Pennec, and Nicholas Ayache. Log-Euclidean metrics for fast and simple calculus on diffusion tensors. *Magnetic Resonance in Medicine*, 56(2):411–421, August 2006.
4. Vincent Arsigny, Pierre Fillard, Xavier Pennec, and Nicholas Ayache. Geometric means in a novel vector space structure on symmetric positive-definite matrices. *SIAM Journal on Matrix Analysis and Applications*, 2007. In press.
5. S. Hun Cheng, N. J. Higham, C. S. Kenney, and A. J. Laub. Approximating the logarithm of a matrix to specified accuracy. *SIAM J. Matrix Anal. Appl.*, 22(4):1112–1125, 2001.
6. Olivier Commowick, Vincent Arsigny, Jimena Costa, Nicholas Ayache, and Grégoire Malandain. An efficient locally affine framework for the registration of anatomical structures. In *Proceedings of the Third IEEE International Symposium on Biomedical Imaging (ISBI 2006)*, pages 478–481, Crystal Gateway Marriott, Arlington, Virginia, USA, April 2006.
7. Olivier Commowick, Vincent Arsigny, Jimena Costa, Nicholas Ayache, and Grégoire Malandain. An efficient locally affine framework for the registration of anatomical structures. *Medical Image Analysis*, 12(4):427–441, 2008.
8. A. Cuzol, P. Hellier, and E. Mémin. A novel parametric method for non-rigid image registration. In G. Christensen and M. Sonka, editors, *Proc. of IPMI'05*, number 3565 in LNCS, pages 456–467, Glenwood Springes, Colorado, USA, July 2005.
9. Jean Gallier. Logarithms and square roots of real matrices. arXiv:0805.0245v1, 2008.
10. N. J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM J. Matrix Anal. Appl.*, 26(4):1179–1193, 2005.
11. J. D. Lambert. *Numerical methods for ordinary differential systems: the initial value problem*. John Wiley & Sons, Inc., New York, NY, USA, 1991.
12. J.A. Little, D.L.G. Hill, and D.J. Hawkes. Deformations incorpotationg rigid structures. *Comp. Vis. and Imag. Under.*, 66(2):223–232, May 1996.
13. J. B. A. Maintz and M. A. Viergever. A survey of medical registration. *Medical image analysis*, 2(1):1–36, 1998.

14. Cleve Moler and Charles Van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM jour. of Matr. Anal. and Appl.*, 20(4):801–836, October 1978.
15. R. Narayanan, J. A. Fessler, H. Park, and C. R. Meyer. Diffeomorphic nonlinear transformations: A local parametric approach for image registration. In *Proceedings of IPMI'05*, volume LNCS 3565, pages 174–185, 2005.
16. X. Papademetris, D. P. Dione, L. W. Dobrucki, L. H. Staib, and A. J. Sinusas. Articulated rigid registration for serial lower-limb mouse imaging. In *Porc. of MICCAI'05 (part 2)*, LNCS 3750, pages 919–926, 2005.
17. Xavier Pennec, Pierre Fillard, and Nicholas Ayache. A Riemannian framework for tensor computing. *International Journal of Computer Vision*, 66(1):41–66, January 2006. A preliminary version appeared as INRIA Research Report 5255, July 2004.
18. Alain Pitiot, Eric Bardinet, Paul M. Thompson, and Grégoire Malandain. Piecewise affine registration of biological images for volume reconstruction. *Medical Image Analysis*, 10(3):465–483, June 2006.
19. D. Rueckert, L. I. Sonoda, C. Hayes, D. L. G. Hill, M. O. Leach, and D. J. Hawkes. Non-rigid registration using free-form deformations: Application to breast MR images. *IEEE Trans. Medecal Imaging*, 18(8):712–721, 1999.
20. D. Sheppard. A two-dimensionnal interpolation function for irregularly spaced data. In *23rd National Conference of the ACM*, pages 517–524, 1968.
21. R. Stefanescu, X. Pennec, and N. Ayache. Grid powered nonlinear image registration with locally adaptive regularization. *Medical Imagage Analalysis*, 8(3):325–342, September 2004.
22. Shlomo Sternberg. *Lectures on Differential Geometry*. Prentice Hall Mathematics Series. Prentice Hall Inc., 1964.
23. M. Tenenbaum and H. Pollard. *Ordinary Differential Equations*. Dover, 1985.